1 An Introduction to Conditional Random Fields for Relational Learning

Charles Sutton
Department of Computer Science
University of Massachusetts, USA
casutton@cs.umass.edu
http://www.cs.umass.edu/∼casutton

Andrew McCallum
Department of Computer Science
University of Massachusetts, USA
mccallum@cs.umass.edu
http://www.cs.umass.edu/∼mccallum

1.1 Introduction

Relational data has two characteristics: first, statistical dependencies exist between the entities we wish to model, and second, each entity often has a rich set of features that can aid classification. For example, when classifying Web documents, the page's text provides much information about the class label, but hyperlinks define a relationship between pages that can improve classification [Taskar et al., 2002]. Graphical models are a natural formalism for exploiting the dependence structure among entities. Traditionally, graphical models have been used to represent the joint probability distribution $p(\mathbf{y}, \mathbf{x})$, where the variables $\mathbf{y}$ represent the attributes of the entities that we wish to predict, and the input variables $\mathbf{x}$ represent our observed knowledge about the entities. But modeling the joint distribution can lead to difficulties when using the rich local features that can occur in relational data, because it requires modeling the distribution $p(\mathbf{x})$, which can include complex dependencies. Modeling these dependencies among inputs can lead to intractable models, but ignoring them can lead to reduced performance.

A solution to this problem is to directly model the conditional distribution $p(\mathbf{y}|\mathbf{x})$, which is sufficient for classification. This is the approach taken by *conditional random fields* [Lafferty et al., 2001]. A conditional random field is simply a conditional distribution $p(\mathbf{y}|\mathbf{x})$ with an associated graphical structure. Because the model is

conditional, dependencies among the input variables $\mathbf{x}$ do not need to be explicitly represented, affording the use of rich, global features of the input. For example, in natural language tasks, useful features include neighboring words and word bigrams, prefixes and suffixes, capitalization, membership in domain-specific lexicons, and semantic information from sources such as WordNet. Recently there has been an explosion of interest in CRFs, with successful applications including text processing [Taskar et al., 2002, Peng and McCallum, 2004, Settles, 2005, Sha and Pereira, 2003], bioinformatics [Sato and Sakakibara, 2005, Liu et al., 2005], and computer vision [He et al., 2004, Kumar and Hebert, 2003].

This chapter is divided into two parts. First, we present a tutorial on current training and inference techniques for conditional random fields. We discuss the important special case of linear-chain CRFs, and then we generalize these to arbitrary graphical structures. We include a brief discussion of techniques for practical CRF implementations.

Second, we present an example of applying a general CRF to a practical relational learning problem. In particular, we discuss the problem of *information extraction*, that is, automatically building a relational database from information contained in unstructured text. Unlike linear-chain models, general CRFs can capture long distance dependencies between labels. For example, if the same name is mentioned more than once in a document, all mentions probably have the same label, and it is useful to extract them all, because each mention may contain different complementary information about the underlying entity. To represent these long-distance dependencies, we propose a *skip-chain CRF*, a model that jointly performs segmentation and collective labeling of extracted mentions. On a standard problem of extracting speaker names from seminar announcements, the skip-chain CRF has better performance than a linear-chain CRF.

1.2 Graphical Models

1.2.1 Definitions

We consider probability distributions over sets of random variables $V = X \cup Y$, where X is a set of *input variables* that we assume are observed, and Y is a set of *output variables* that we wish to predict. Every variable $v \in V$ takes outcomes from a set $\mathcal{V}$, which can be either continuous or discrete, although we discuss only the discrete case in this chapter. We denote an assignment to X by $\mathbf{x}$, and we denote an assignment to a set $A \subset X$ by $\mathbf{x}_A$, and similarly for Y. We use the notation $\mathbf{1}_{\{x=x'\}}$ to denote an indicator function of x which takes the value 1 when $x = x'$ and 0 otherwise.

A graphical model is a family of probability distributions that factorize according to an underlying graph. The main idea is to represent a distribution over a large number of random variables by a product of local functions that each depend on only a small number of variables. Given a collection of subsets $A \subset V$, we define

an *undirected graphical model* as the set of all distributions that can be written in the form

constraints — finite support!

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_A \Psi_A(\mathbf{x}_A, \mathbf{y}_A), \tag{1.1}$$

for any choice of *factors* $F = \{\Psi_A\}$, where $\Psi_A : \mathcal{V}^n \to \Re^+$. (These functions are also called *local functions* or *compatibility functions*.) We will occasionally use the term *random field* to refer to a particular distribution among those defined by an undirected model. To reiterate, we will consistently use the term *model* to refer to a family of distributions, and *random field* (or more commonly, distribution) to refer to a single one.

The constant Z is a normalization factor defined as

$$Z = \sum_{\mathbf{x},\mathbf{y}} \prod_A \Psi_A(\mathbf{x}_A, \mathbf{y}_A), \tag{1.2}$$

which ensures that the distribution sums to 1. The quantity Z, considered as a function of the set F of factors, is called the *partition function* in the statistical physics and graphical models communities. Computing Z is intractable in general, but much work exists on how to approximate it.

Graphically, we represent the factorization (1.1) by a *factor graph* [Kschischang et al., 2001]. A factor graph is a bipartite graph $G = (V, F, E)$ in which a variable node $v_s \in V$ is connected to a factor node $\Psi_A \in F$ if v_s is an argument to Ψ_A. An example of a factor graph is shown graphically in Figure 1.1 (right). In that figure, the circles are variable nodes, and the shaded boxes are factor nodes.

In this chapter, we will assume that each local function has the form

$$\Psi_A(\mathbf{x}_A, \mathbf{y}_A) = \exp\left\{\sum_k \theta_{Ak} f_{Ak}(\mathbf{x}_A, \mathbf{y}_A)\right\}, \tag{1.3}$$

for some real-valued parameter vector θ_A, and for some set of *feature functions* or *sufficient statistics* $\{f_{Ak}\}$. This form ensures that the family of distributions over V parameterized by θ is an exponential family. Much of the discussion in this chapter actually applies to exponential families in general.

A *directed graphical model*, also known as a Bayesian network, is based on a directed graph $G = (V, E)$. A directed model is a family of distributions that factorize as:

$$p(\mathbf{y}, \mathbf{x}) = \prod_{v \in V} p(v|\pi(v)), \tag{1.4}$$

where $\pi(v)$ are the parents of v in G. An example of a directed model is shown in Figure 1.1 (left).

We use the term *generative model* to refer to a directed graphical model in which the outputs topologically precede the inputs, that is, no $x \in X$ can be a parent of an output $y \in Y$. Essentially, a generative model is one that directly describes how the outputs probabilistically "generate" the inputs.

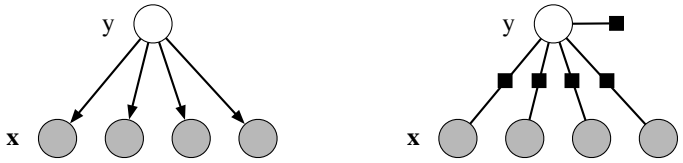

Figure 1.1 The naive Bayes classifier, as a directed model (left), and as a factor graph (right).

1.2.2 Applications of graphical models

In this section we discuss a few applications of graphical models to natural language processing. Although these examples are well-known, they serve both to clarify the definitions in the previous section, and to illustrate some ideas that will arise again in our discussion of conditional random fields. We devote special attention to the hidden Markov model (HMM), because it is closely related to the linear-chain CRF.

1.2.2.1 Classification

First we discuss the problem of *classification*, that is, predicting a single class variable y given a vector of features $\mathbf{x} = (x_1, x_2, \ldots, x_K)$. One simple way to accomplish this is to assume that once the class label is known, all the features are independent. The resulting classifier is called the *naive Bayes classifier*. It is based on a joint probability model of the form:

$$p(y, \mathbf{x}) = p(y) \prod_{k=1}^{K} p(x_k|y). \tag{1.5}$$

This model can be described by the directed model shown in Figure 1.1 (left). We can also write this model as a factor graph, by defining a factor $\Psi(y) = p(y)$, and a factor $\Psi_k(y, x_k) = p(x_k|y)$ for each feature x_k. This factor graph is shown in Figure 1.1 (right).

Another well-known classifier that is naturally represented as a graphical model is logistic regression (sometimes known as the *maximum entropy classifier* in the NLP community). In statistics, this classifier is motivated by the assumption that the log probability, $\log p(y|\mathbf{x})$, of each class is a linear function of $\mathbf{x}$, plus a normalization constant. This leads to the conditional distribution:

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\lambda_y + \sum_{j=1}^{K} \lambda_{y,j} x_j\right\}, \tag{1.6}$$

where $Z(\mathbf{x}) = \sum_y \exp\{\lambda_y + \sum_{j=1}^{K} \lambda_{y,j} x_j\}$ is a normalizing constant, and λ_y is a bias weight that acts like $\log p(y)$ in naive Bayes. Rather than using one vector per class, as in (1.6), we can use a different notation in which a single set of weights is shared across all the classes. The trick is to define a set of *feature functions* that are

nonzero only for a single class. To do this, the feature functions can be defined as $f_{y',j}(y, \mathbf{x}) = \mathbf{1}_{\{y'=y\}} x_j$ for the feature weights and $f_{y'}(y, \mathbf{x}) = \mathbf{1}_{\{y'=y\}}$ for the bias weights. Now we can use f_k to index each feature function $f_{y',j}$, and λ_k to index its corresponding weight $\lambda_{y',j}$. Using this notational trick, the logistic regression model becomes:

$$p(y|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y, \mathbf{x})\right\}. \tag{1.7}$$

We introduce this notation because it mirrors the usual notation for conditional random fields.

1.2.2.2 Sequence Models

Classifiers predict only a single class variable, but the true power of graphical models lies in their ability to model many variables that are interdependent. In this section, we discuss perhaps the simplest form of dependency, in which the output variables are arranged in a sequence. To motivate this kind of model, we discuss an application from natural language processing, the task of *named-entity recognition* (NER). NER is the problem of identifying and classifying proper names in text, including locations, such as *China*; people, such as *George Bush*; and organizations, such as the *United Nations*. The named-entity recognition task is, given a sentence, first to segment which words are part of entities, and then to classify each entity by type (person, organization, location, and so on). The challenge of this problem is that many named entities are too rare to appear even in a large training set, and therefore the system must identify them based only on context.

One approach to NER is to classify each word independently as one of either PERSON, LOCATION, ORGANIZATION, or OTHER (meaning not an entity). The problem with this approach is that it assumes that given the input, all of the named-entity labels are independent. In fact, the named-entity labels of neighboring words are dependent; for example, while *New York* is a location, *New York Times* is an organization.

This independence assumption can be relaxed by arranging the output variables in a linear chain. This is the approach taken by the hidden Markov model (HMM) [Rabiner, 1989]. An HMM models a sequence of observations $X = \{x_t\}_{t=1}^{\mathrm{T}}$ by assuming that there is an underlying sequence of *states* $Y = \{y_t\}_{t=1}^{\mathrm{T}}$ drawn from a finite state set S. In the named-entity example, each observation x_t is the identity of the word at position t, and each state y_t is the named-entity label, that is, one of the entity types PERSON, LOCATION, ORGANIZATION, and OTHER.

To model the joint distribution $p(\mathbf{y}, \mathbf{x})$ tractably, an HMM makes two independence assumptions. First, it assumes that each state depends only on its immediate predecessor, that is, each state y_t is independent of all its ancestors $y_1, y_2, \ldots, y_{t-2}$ given its previous state y_{t-1}. Second, an HMM assumes that each observation variable x_t depends only on the current state y_t. With these assumptions, we can

specify an HMM using three probability distributions: first, the distribution $p(y_1)$ over initial states; second, the transition distribution $p(y_t|y_{t-1})$; and finally, the observation distribution $p(x_t|y_t)$. That is, the joint probability of a state sequence $\mathbf{y}$ and an observation sequence $\mathbf{x}$ factorizes as

$$p(\mathbf{y}, \mathbf{x}) = \prod_{t=1}^{\mathrm{T}} p(y_t|y_{t-1})p(x_t|y_t), \tag{1.8}$$

where, to simplify notation, we write the initial state distribution $p(y_1)$ as $p(y_1|y_0)$. In natural language processing, HMMs have been used for sequence labeling tasks such as part-of-speech tagging, named-entity recognition, and information extraction.

1.2.3 Discriminative and Generative Models

An important difference between naive Bayes and logistic regression is that naive Bayes is *generative*, meaning that it is based on a model of the joint distribution $p(y, \mathbf{x})$, while logistic regression is *discriminative*, meaning that it is based on a model of the conditional distribution $p(y|\mathbf{x})$. In this section, we discuss the differences between generative and discriminative modeling, and the advantages of discriminative modeling for many tasks. For concreteness, we focus on the examples of naive Bayes and logistic regression, but the discussion in this section actually applies in general to the differences between generative models and conditional random fields.

The main difference is that a conditional distribution $p(\mathbf{y}|\mathbf{x})$ does not include a model of $p(\mathbf{x})$, which is not needed for classification anyway. The difficulty in modeling $p(\mathbf{x})$ is that it often contains many highly dependent features, which are difficult to model. For example, in named-entity recognition, an HMM relies on only one feature, the word's identity. But many words, especially proper names, will not have occurred in the training set, so the word-identity feature is uninformative. To label unseen words, we would like to exploit other features of a word, such as its capitalization, its neighboring words, its prefixes and suffixes, its membership in predetermined lists of people and locations, and so on.

To include interdependent features in a generative model, we have two choices: enhance the model to represent dependencies among the inputs, or make simplifying independence assumptions, such as the naive Bayes assumption. The first approach, enhancing the model, is often difficult to do while retaining tractability. For example, it is hard to imagine how to model the dependence between the capitalization of a word and its suffixes, nor do we particularly wish to do so, since we always observe the test sentences anyway. The second approach, adding independence assumptions among the inputs, is problematic because it can hurt performance. For example, although the naive Bayes classifier performs surprisingly well in document classification, it performs worse on average across a range of applications than logistic regression [Caruana and Niculescu-Mizil, 2005].

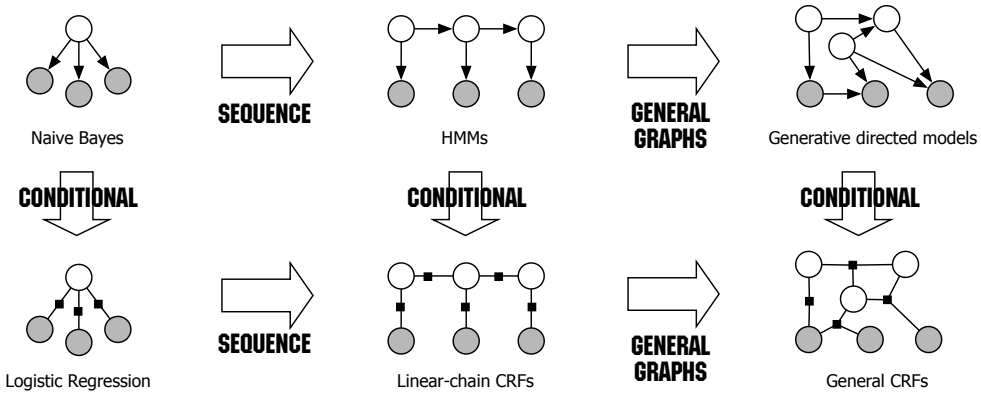

Figure 1.2 Diagram of the relationship between naive Bayes, logistic regression, HMMs, linear-chain CRFs, generative models, and general CRFs.

Furthermore, even when naive Bayes has good classification accuracy, its probability estimates tend to be poor. To understand why, imagine training naive Bayes on a data set in which all the features are repeated, that is, $\mathbf{x} = (x_1, x_1, x_2, x_2, \ldots, x_K, x_K)$. This will increase the confidence of the naive Bayes probability estimates, even though no new information has been added to the data. Assumptions like naive Bayes can be especially problematic when we generalize to sequence models, because inference essentially combines evidence from different parts of the model. If probability estimates at a local level are overconfident, it might be difficult to combine them sensibly.

Actually, the difference in performance between naive Bayes and logistic regression is due *only* to the fact that the first is generative and the second discriminative; the two classifiers are, for discrete input, identical in all other respects. Naive Bayes and logistic regression consider the same hypothesis space, in the sense that any logistic regression classifier can be converted into a naive Bayes classifier with the same decision boundary, and vice versa. Another way of saying this is that the naive Bayes model (1.5) defines the same family of distributions as the logistic regression model (1.7), if we interpret it generatively as

$$p(y, \mathbf{x}) = \frac{\exp\left\{\sum_k \lambda_k f_k(y, \mathbf{x})\right\}}{\sum_{\tilde{y}, \tilde{\mathbf{x}}} \exp\left\{\sum_k \lambda_k f_k(\tilde{y}, \tilde{\mathbf{x}})\right\}}. \tag{1.9}$$

This means that if the naive Bayes model (1.5) is trained to maximize the conditional likelihood, we recover the same classifier as from logistic regression. Conversely, if the logistic regression model is interpreted generatively, as in (1.9), and is trained to maximize the joint likelihood $p(y, \mathbf{x})$, then we recover the same classifier as from naive Bayes. In the terminology of Ng and Jordan [2002], naive Bayes and logistic regression form a *generative-discriminative pair*.

The principal advantage of discriminative modeling is that it is better suited to

including rich, overlapping features. To understand this, consider the family of naive Bayes distributions (1.5). This is a family of joint distributions whose conditionals all take the "logistic regression form" (1.7). But there are many other joint models, some with complex dependencies among $\mathbf{x}$, whose conditional distributions also have the form (1.7). By modeling the conditional distribution directly, we can remain agnostic about the form of $p(\mathbf{x})$. This may explain why it has been observed that conditional random fields tend to be more robust than generative models to violations of their independence assumptions [Lafferty et al., 2001]. Simply put, CRFs make independence assumptions among $\mathbf{y}$, but not among $\mathbf{x}$.

Another way to make the same point is due to Minka [2005]. Suppose we have a generative model p_g with parameters θ. By definition, this takes the form

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{y}; \theta) p_g(\mathbf{x}|\mathbf{y}; \theta). \tag{1.10}$$

But we could also rewrite p_g using Bayes rule as

$$p_g(\mathbf{y}, \mathbf{x}; \theta) = p_g(\mathbf{x}; \theta) p_g(\mathbf{y}|\mathbf{x}; \theta), \tag{1.11}$$

where $p_g(\mathbf{x}; \theta)$ and $p_g(\mathbf{y}|\mathbf{x}; \theta)$ are computed by inference, i.e., $p_g(\mathbf{x}; \theta) = \sum_{\mathbf{y}} p_g(\mathbf{y}, \mathbf{x}; \theta)$ and $p_g(\mathbf{y}|\mathbf{x}; \theta) = p_g(\mathbf{y}, \mathbf{x}; \theta)/p_g(\mathbf{x}; \theta)$.

Now, compare this generative model to a discriminative model over the same family of joint distributions. To do this, we define a prior $p(\mathbf{x})$ over inputs, such that $p(\mathbf{x})$ could have arisen from p_g with some parameter setting. That is, $p(\mathbf{x}) = p_c(\mathbf{x}; \theta') = \sum_{\mathbf{y}} p_g(\mathbf{y}, \mathbf{x}|\theta')$. We combine this with a conditional distribution $p_c(\mathbf{y}|\mathbf{x}; \theta)$ that could also have arisen from p_g, that is, $p_c(\mathbf{y}|\mathbf{x}; \theta) = p_g(\mathbf{y}, \mathbf{x}; \theta)/p_g(\mathbf{x}; \theta)$. Then the resulting distribution is

$$p_c(\mathbf{y}, \mathbf{x}) = p_c(\mathbf{x}; \theta') p_c(\mathbf{y}|\mathbf{x}; \theta). \tag{1.12}$$

By comparing (1.11) with (1.12), it can be seen that the conditional approach has more freedom to fit the data, because it does not require that $\theta = \theta'$. Intuitively, because the parameters θ in (1.11) are used in both the input distribution and the conditional, a good set of parameters must represent both well, potentially at the cost of trading off accuracy on $p(\mathbf{y}|\mathbf{x})$, the distribution we care about, for accuracy on $p(\mathbf{x})$, which we care less about.

In this section, we have discussed the relationship between naive Bayes and logistic regression in detail because it mirrors the relationship between HMMs and linear-chain CRFs. Just as naive Bayes and logistic regression are a generative-discriminative pair, there is a discriminative analog to hidden Markov models, and this analog is a particular type of conditional random field, as we explain next. The analogy between naive Bayes, logistic regression, generative models, and conditional random fields is depicted in Figure 1.2.

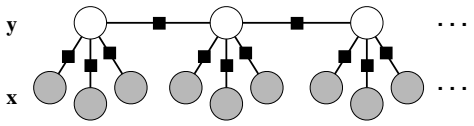

Figure 1.3 Graphical model of an HMM-like linear-chain CRF.

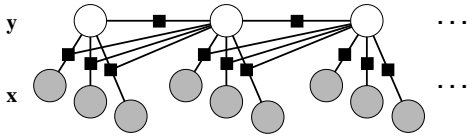

Figure 1.4 Graphical model of a linear-chain CRF in which the transition score depends on the current observation.

1.3 Linear-Chain Conditional Random Fields

In the previous section, we have seen advantages both to discriminative modeling and sequence modeling. So it makes sense to combine the two. This yields a linear-chain CRF, which we describe in this section. First, in Section 1.3.1, we define linear-chain CRFs, motivating them from HMMs. Then, we discuss parameter estimation (Section 1.3.2) and inference (Section 1.3.3) in linear-chain CRFs.

1.3.1 From HMMs to CRFs

To motivate our introduction of linear-chain conditional random fields, we begin by considering the conditional distribution $p(\mathbf{y}|\mathbf{x})$ that follows from the joint distribution $p(\mathbf{y}, \mathbf{x})$ of an HMM. The key point is that this conditional distribution is in fact a conditional random field with a particular choice of feature functions. First, we rewrite the HMM joint (1.8) in a form that is more amenable to generalization. This is

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp \left\{ \sum_t \sum_{i,j \in S} \lambda_{ij} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{y_{t-1} = j\}} + \sum_t \sum_{i \in S} \sum_{o \in O} \mu_{oi} \mathbf{1}_{\{y_t = i\}} \mathbf{1}_{\{x_t = o\}} \right\}, \tag{1.13}$$

where $\theta = \{\lambda_{ij}, \mu_{oi}\}$ are the parameters of the distribution, and can be any real numbers. Every HMM can be written in this form, as can be seen simply by setting $\lambda_{ij} = \log p(y' = i | y = j)$ and so on. Because we do not require the parameters to be log probabilities, we are no longer guaranteed that the distribution sums to 1, unless we explicitly enforce this by using a normalization constant Z. Despite this added flexibility, it can be shown that (1.13) describes exactly the class of HMMs in (1.8); we have added flexibility to the parameterization, but we have not added any distributions to the family.

We can write (1.13) more compactly by introducing the concept of *feature functions*, just as we did for logistic regression in (1.7). Each feature function has the form $f_k(y_t, y_{t-1}, x_t)$. In order to duplicate (1.13), there needs to be one feature $f_{ij}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{y'=j\}}$ for each transition (i, j) and one feature $f_{io}(y, y', x) = \mathbf{1}_{\{y=i\}}\mathbf{1}_{\{x=o\}}$ for each state-observation pair (i, o). Then we can write an HMM as:

$$p(\mathbf{y}, \mathbf{x}) = \frac{1}{Z} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\}. \tag{1.14}$$

Again, equation (1.14) defines exactly the same family of distributions as (1.13), and therefore as the original HMM equation (1.8).
The last step is to write the conditional distribution $p(\mathbf{y}|\mathbf{x})$ that results from the HMM (1.14). This is

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{y}, \mathbf{x})}{\sum_{\mathbf{y}'} p(\mathbf{y}', \mathbf{x})} = \frac{\exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, x_t)\right\}}{\sum_{\mathbf{y}'} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y'_t, y'_{t-1}, x_t)\right\}}. \tag{1.15}$$

This conditional distribution (1.15) is a linear-chain CRF, in particular one that includes features only for the current word's identity. But many other linear-chain CRFs use richer features of the input, such as prefixes and suffixes of the current word, the identity of surrounding words, and so on. Fortunately, this extension requires little change to our existing notation. We simply allow the feature functions $f_k(y_t, y_{t-1}, \mathbf{x}_t)$ to be more general than indicator functions. This leads to the general definition of linear-chain CRFs, which we present now.

Definition 1.1
Let Y, X be random vectors, $\Lambda = \{\lambda_k\} \in \Re^K$ be a parameter vector, and $\{f_k(y, y', \mathbf{x}_t)\}_{k=1}^{K}$ be a set of real-valued feature functions. Then a *linear-chain conditional random field* is a distribution $p(\mathbf{y}|\mathbf{x})$ that takes the form

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}, \tag{1.16}$$

where $Z(\mathbf{x})$ is an instance-specific normalization function

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \exp\left\{\sum_{k=1}^{K} \lambda_k f_k(y_t, y_{t-1}, \mathbf{x}_t)\right\}. \tag{1.17}$$

We have just seen that if the joint $p(\mathbf{y}, \mathbf{x})$ factorizes as an HMM, then the associated conditional distribution $p(\mathbf{y}|\mathbf{x})$ is a linear-chain CRF. This HMM-like CRF is pictured in Figure 1.3. Other types of linear-chain CRFs are also useful, however. For example, in an HMM, a transition from state i to state j receives the same score, $\log p(y_t = j|y_{t-1} = i)$, regardless of the input. In a CRF, we can allow the score of the transition (i, j) to depend on the current observation vector, simply

by adding a feature $\mathbf{1}_{\{y_t=j\}}\mathbf{1}_{\{y_{t-1}=1\}}\mathbf{1}_{\{x_t=o\}}$. A CRF with this kind of transition feature, which is commonly used in text applications, is pictured in Figure 1.4.
To indicate in the definition of linear-chain CRF that each feature function can depend on observations from any time step, we have written the observation argument to f_k as a vector $\mathbf{x}_t$, which should be understood as containing all the components of the global observations $\mathbf{x}$ that are needed for computing features at time t. For example, if the CRF uses the next word x_{t+1} as a feature, then the feature vector $\mathbf{x}_t$ is assumed to include the identity of word x_{t+1}.
Finally, note that the normalization constant $Z(\mathbf{x})$ sums over all possible state sequences, an exponentially large number of terms. Nevertheless, it can be computed efficiently by forward-backward, as we explain in Section 1.3.3.

1.3.2 Parameter Estimation

In this section we discuss how to estimate the parameters $\theta = \{\lambda_k\}$ of a linear-chain CRF. We are given iid training data $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N$, where each $\mathbf{x}^{(i)} = \{x_1^{(i)}, x_2^{(i)}, \ldots x_T^{(i)}\}$ is a sequence of inputs, and each $\mathbf{y}^{(i)} = \{y_1^{(i)}, y_2^{(i)}, \ldots y_T^{(i)}\}$ is a sequence of the desired predictions. Thus, we have relaxed the iid assumption within each sequence, but we still assume that distinct sequences are independent. (In Section 1.4, we will see how to relax this assumption as well.)
Parameter estimation is typically performed by penalized maximum likelihood. Because we are modeling the conditional distribution, the following log likelihood, sometimes called the *conditional log likelihood*, is appropriate:

$$\ell(\theta) = \sum_{i=1}^{N} \log p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)}). \tag{1.18}$$

One way to understand the conditional likelihood $p(\mathbf{y}|\mathbf{x};\theta)$ is to imagine combining it with some arbitrary prior $p(\mathbf{x};\theta')$ to form a joint $p(\mathbf{y},\mathbf{x})$. Then when we optimize the joint log likelihood

$$\log p(\mathbf{y},\mathbf{x}) = \log p(\mathbf{y}|\mathbf{x};\theta) + \log p(\mathbf{x};\theta'), \tag{1.19}$$

the two terms on the right-hand side are decoupled, that is, the value of θ' does not affect the optimization over θ. If we do not need to estimate $p(\mathbf{x})$, then we can simply drop the second term, which leaves (1.18).
After substituting in the CRF model (1.16) into the likelihood (1.18), we get the following expression:

$$\ell(\theta) = \sum_{i=1}^{N}\sum_{t=1}^{T}\sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}), \tag{1.20}$$

Before we discuss how to optimize this, we mention regularization. It is often the case that we have a large number of parameters. As a measure to avoid overfitting, we use *regularization*, which is a penalty on weight vectors whose norm is too

large. A common choice of penalty is based on the Euclidean norm of θ and on a *regularization parameter* $1/2\sigma^2$ that determines the strength of the penalty. Then the regularized log likelihood is

$$\ell(\theta) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} \lambda_k f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \log Z(\mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k^2}{2\sigma^2}. \tag{1.21}$$

The notation for the regularizer is intended to suggest that regularization can also be viewed as performing maximum a posteriori estimation of θ, if θ is assigned a Gaussian prior with mean 0 and covariance $\sigma^2 I$. The parameter σ^2 is a free parameter which determines how much to penalize large weights. Determining the best regularization parameter can require a computationally-intensive parameter sweep. Fortunately, often the accuracy of the final model does not appear to be sensitive to changes in σ^2, even when σ^2 is varied up to a factor of 10. An alternative choice of regularization is to use the ℓ_1 norm instead of the Euclidean norm, which corresponds to an exponential prior on parameters [Goodman, 2004]. This regularizer tends to encourage sparsity in the learned parameters.

In general, the function $\ell(\theta)$ cannot be maximized in closed form, so numerical optimization is used. The partial derivatives of (1.21) are

$$\frac{\partial \ell}{\partial \lambda_k} = \sum_{i=1}^{N} \sum_{t=1}^{T} f_k(y_t^{(i)}, y_{t-1}^{(i)}, \mathbf{x}_t^{(i)}) - \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{y,y'} f_k(y, y', \mathbf{x}_t^{(i)}) p(y, y' | \mathbf{x}^{(i)}) - \sum_{k=1}^{K} \frac{\lambda_k}{\sigma^2}. \tag{1.22}$$

The first term is the expected value of f_k under the empirical distribution:

$$\tilde{p}(\mathbf{y}, \mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\{\mathbf{y}=\mathbf{y}^{(i)}\}} \mathbf{1}_{\{\mathbf{x}=\mathbf{x}^{(i)}\}}. \tag{1.23}$$

The second term, which arises from the derivative of $\log Z(\mathbf{x})$, is the expectation of f_k under the model distribution $p(\mathbf{y}|\mathbf{x};\theta)\tilde{p}(\mathbf{x})$. Therefore, at the unregularized maximum likelihood solution, when the gradient is zero, these two expectations are equal. This pleasing interpretation is a standard result about maximum likelihood estimation in exponential families.

Now we discuss how to optimize $\ell(\theta)$. The function $\ell(\theta)$ is concave, which follows from the convexity of functions of the form $g(\mathbf{x}) = \log \sum_i \exp x_i$. Convexity is extremely helpful for parameter estimation, because it means that every local optimum is also a global optimum. Adding regularization ensures that ℓ is strictly concave, which implies that it has exactly one global optimum.

Perhaps the simplest approach to optimize ℓ is steepest ascent along the gradient (1.22), but this requires too many iterations to be practical. Newton's method converges much faster because it takes into account the curvature of the likelihood, but it requires computing the Hessian, the matrix of all second derivatives. The size of the Hessian is quadratic in the number of parameters. Since practical applications often use tens of thousands or even millions of parameters, even storing the full Hessian is not practical.

Instead, current techniques for optimizing (1.21) make approximate use of second-order information. Particularly successful have been quasi-Newton methods such as BFGS [Bertsekas, 1999], which compute an approximation to the Hessian from only the first derivative of the objective function. A full $K \times K$ approximation to the Hessian still requires quadratic size, however, so a limited-memory version of BFGS is used, due to Byrd et al. [1994]. As an alternative to limited-memory BFGS, conjugate gradient is another optimization technique that also makes approximate use of second-order information and has been used successfully with CRFs. Either can be thought of as a black-box optimization routine that is a drop-in replacement for vanilla gradient ascent. When such second-order methods are used, gradient-based optimization is much faster than the original approaches based on iterative scaling in Lafferty et al. [2001], as shown experimentally by several authors [Sha and Pereira, 2003, Wallach, 2002, Malouf, 2002, Minka, 2003].

Finally, it is important to remark on the computational cost of training. Both the partition function $Z(\mathbf{x})$ in the likelihood and the marginal distributions $p(y_t, y_{t-1}|\mathbf{x})$ in the gradient can be computed by forward-backward, which uses computational complexity $O(TM^2)$. However, each training instance will have a different partition function and marginals, so we need to run forward-backward for each training instance for each gradient computation, for a total training cost of $O(TM^2NG)$, where N is the number of training examples, and G the number of gradient computations required by the optimization procedure. For many data sets, this cost is reasonable, but if the number of states is large, or the number of training sequences is very large, then this can become expensive. For example, on a standard named-entity data set, with 11 labels and 200,000 words of training data, CRF training finishes in under two hours on current hardware. However, on a part-of-speech tagging data set, with 45 labels and one million words of training data, CRF training requires over a week.

1.3.3 Inference

There are two common inference problems for CRFs. First, during training, computing the gradient requires marginal distributions for each edge $p(y_t, y_{t-1}|\mathbf{x})$, and computing the likelihood requires $Z(\mathbf{x})$. Second, to label an unseen instance, we compute the most likely (Viterbi) labeling $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. In linear-chain CRFs, both inference tasks can be performed efficiently and exactly by variants of the standard dynamic-programming algorithms for HMMs. In this section, we briefly review the HMM algorithms, and extend them to linear-chain CRFs. These standard inference algorithms are described in more detail by Rabiner [1989].

First, we introduce notation which will simplify the forward-backward recursions. An HMM can be viewed as a factor graph $p(\mathbf{y}, \mathbf{x}) = \prod_t \Psi_t(y_t, y_{t-1}, x_t)$ where $Z = 1$, and the factors are defined as:

$$\Psi_t(j, i, x) \stackrel{\text{def}}{=} p(y_t = j|y_{t-1} = i)p(x_t = x|y_t = j). \tag{1.24}$$

If the HMM is viewed as a weighted finite state machine, then $\Psi_t(j, i, x)$ is the weight on the transition from state i to state j when the current observation is x. Now, we review the HMM forward algorithm, which is used to compute the probability $p(\mathbf{x})$ of the observations. The idea behind forward-backward is to first rewrite the naive summation $p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}, \mathbf{y})$ using the distributive law:

$$p(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, x_t) \tag{1.25}$$

$$= \sum_{y_{\mathrm{T}}} \sum_{y_{\mathrm{T}-1}} \Psi_{\mathrm{T}}(y_{\mathrm{T}}, y_{\mathrm{T}-1}, x_{\mathrm{T}}) \sum_{y_{\mathrm{T}-2}} \Psi_{\mathrm{T}-1}(y_{\mathrm{T}-1}, y_{\mathrm{T}-2}, x_{\mathrm{T}-1}) \sum_{y_{\mathrm{T}-3}} \cdots \tag{1.26}$$

Now we observe that each of the intermediate sums is reused many times during the computation of the outer sum, and so we can save an exponential amount of work by caching the inner sums.

This leads to defining a set of *forward variables* α_t, each of which is a vector of size M (where M is the number of states) which stores one of the intermediate sums. These are defined as:

$$\alpha_t(j) \stackrel{\text{def}}{=} p(\mathbf{x}_{\langle 1 \ldots t \rangle}, y_t = j) \tag{1.27}$$

$$= \sum_{\mathbf{y}_{\langle 1 \ldots t-1 \rangle}} \Psi_t(j, y_{t-1}, x_t) \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}), \tag{1.28}$$

where the summation over $\mathbf{y}_{\langle 1 \ldots t-1 \rangle}$ ranges over all assignments to the sequence of random variables $y_1, y_2, \ldots, y_{t-1}$. The alpha values can be computed by the recursion

$$\alpha_t(j) = \sum_{i \in S} \Psi_t(j, i, x_t) \alpha_{t-1}(i), \tag{1.29}$$

with initialization $\alpha_1(j) = \Psi_1(j, y_0, x_1)$. (Recall that y_0 is the fixed initial state of the HMM.) It is easy to see that $p(\mathbf{x}) = \sum_{y_{\mathrm{T}}} \alpha_{\mathrm{T}}(y_{\mathrm{T}})$ by repeatedly substituting the recursion (1.29) to obtain (1.26). A formal proof would use induction.

The backward recursion is exactly the same, except that in (1.26), we push in the summations in reverse order. This results in the definition

$$\beta_t(i) \stackrel{\text{def}}{=} p(\mathbf{x}_{\langle t+1 \ldots \mathrm{T} \rangle} | y_t = i) \tag{1.30}$$

$$= \sum_{\mathbf{y}_{\langle t+1 \ldots \mathrm{T} \rangle}} \prod_{t'=t+1}^{\mathrm{T}} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}), \tag{1.31}$$

and the recursion

$$\beta_t(i) = \sum_{j \in S} \Psi_{t+1}(j, i, x_{t+1}) \beta_{t+1}(j), \tag{1.32}$$

which is initialized $\beta_{\mathrm{T}}(i) = 1$. Analogously to the forward case, we can compute $p(\mathbf{x})$ using the backward variables as $p(\mathbf{x}) = \beta_0(y_0) \stackrel{\text{def}}{=} \sum_{y_1} \Psi_1(y_1, y_0, x_1) \beta_1(y_1)$.

By combining results from the forward and backward recursions, we can compute the marginal distributions needed for the gradient (1.22). Applying the distributive law again, we see that

$$p(y_{t-1}, y_t|\mathbf{x}) = \Psi_t(y_t, y_{t-1}, x_t) \left(\sum_{\mathbf{y}_{\langle 1 \dots t-2 \rangle}} \prod_{t'=1}^{t-1} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \right) \left(\sum_{\mathbf{y}_{\langle t+1 \dots \mathrm{T} \rangle}} \prod_{t'=t+1}^{T} \Psi_{t'}(y_{t'}, y_{t'-1}, x_{t'}) \right), \quad (1.33)$$

which can be computed from the forward and backward recursions as

$$p(y_{t-1}, y_t|\mathbf{x}) \propto \alpha_{t-1}(y_{t-1}) \Psi_t(y_t, y_{t-1}, x_t) \beta_t(y_t). \quad (1.34)$$

Finally, to compute the globally most probable assignment $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$, we observe that the trick in (1.26) still works if all the summations are replaced by maximization. This yields the Viterbi recursion:

$$\delta_t(j) = \max_{i \in S} \Psi_t(j, i, x_t) \delta_{t-1}(i) \quad (1.35)$$

Now that we have described the forward-backward and Viterbi algorithms for HMMs, the generalization to linear-chain CRFs is fairly straightforward. The forward-backward algorithm for linear-chain CRFs is identical to the HMM version, except that the transition weights $\Psi_t(j, i, x_t)$ are defined differently. We observe that the CRF model (1.16) can be rewritten as:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x}_t), \quad (1.36)$$

where we define

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}_t) = \exp \left\{ \sum_k \lambda_k f_k(y_t, y_{t-1}, \mathbf{x}_t) \right\}. \quad (1.37)$$

With that definition, the forward recursion (1.29), the backward recursion (1.32), and the Viterbi recursion (1.35) can be used unchanged for linear-chain CRFs. Instead of computing $p(\mathbf{x})$ as in an HMM, in a CRF the forward and backward recursions compute $Z(\mathbf{x})$.

A final inference task that is useful in some applications is to compute a marginal probability $p(y_t, y_{t+1}, \dots y_{t+k}|\mathbf{x})$ over a range of nodes. For example, this is useful for measuring the model's confidence in its predicted labeling over a segment of input. This marginal probability can be computed efficiently using constrained forward-backward, as described by Culotta and McCallum [2004].

1.4 CRFs in General

In this section, we define CRFs with general graphical structure, as they were introduced originally [Lafferty et al., 2001]. Although initial applications of CRFs used linear chains, there have been many later applications of CRFs with more general graphical structures. Such structures are especially useful for relational learning, because they allow relaxing the iid assumption among entities. Also, although CRFs have typically been used for across-network classification, in which the training and testing data are assumed to be independent, we will see that CRFs can be used for within-network classification as well, in which we model probabilistic dependencies between the training and testing data.
The generalization from linear-chain CRFs to general CRFs is fairly straightforward. We simply move from using a linear-chain factor graph to a more general factor graph, and from forward-backward to more general (perhaps approximate) inference algorithms.

1.4.1 Model

First we present the general definition of a conditional random field.

Definition 1.2
Let G be a factor graph over Y. Then $p(\mathbf{y}|\mathbf{x})$ is a conditional random field if for any fixed $\mathbf{x}$, the distribution $p(\mathbf{y}|\mathbf{x})$ factorizes according to G.

Thus, every conditional distribution $p(\mathbf{y}|\mathbf{x})$ is a CRF for some, perhaps trivial, factor graph. If $F = \{\Psi_A\}$ is the set of factors in G, and each factor takes the exponential family form (1.3), then the conditional distribution can be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{\Psi_A \in G} \exp\left\{ \sum_{k=1}^{K(A)} \lambda_{Ak} f_{Ak}(\mathbf{y}_A, \mathbf{x}_A) \right\}. \tag{1.38}$$

In addition, practical models rely extensively on parameter tying. For example, in the linear-chain case, often the same weights are used for the factors $\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)$ at each time step. To denote this, we partition the factors of G into $\mathcal{C} = \{C_1, C_2, \ldots C_P\}$, where each C_p is a *clique template* whose parameters are tied. This notion of clique template generalizes that in Taskar et al. [2002], Sutton et al. [2004], and Richardson and Domingos [2005]. Each clique template C_p is a set of factors which has a corresponding set of sufficient statistics $\{f_{pk}(\mathbf{x}_p, \mathbf{y}_p)\}$ and parameters $\theta_p \in \Re^{K(p)}$. Then the CRF can be written as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p), \tag{1.39}$$

where each factor is parameterized as

$$\Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p) = \exp\left\{\sum_{k=1}^{K(p)} \lambda_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c)\right\}, \tag{1.40}$$

and the normalization function is

$$Z(\mathbf{x}) = \sum_{\mathbf{y}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{y}_c; \theta_p). \tag{1.41}$$

For example, in a linear-chain conditional random field, typically one clique template $C = \{\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)\}_{t=1}^{\mathrm{T}}$ is used for the entire network.

Several special cases of conditional random fields are of particular interest. First, *dynamic conditional random fields* [Sutton et al., 2004] are sequence models which allow multiple labels at each time step, rather than single labels as in linear-chain CRFs. Second, *relational Markov networks* [Taskar et al., 2002] are a type of general CRF in which the graphical structure and parameter tying are determined by an SQL-like syntax. Finally, *Markov logic networks* [Richardson and Domingos, 2005, Singla and Domingos, 2005] are a type of probabilistic logic in which there are parameters for each first-order rule in a knowledge base.

1.4.2 Applications of CRFs

CRFs have been applied to a variety of domains, including text processing, computer vision, and bioinformatics. In this section, we discuss several applications, highlighting the different graphical structures that occur in the literature.

One of the first large-scale applications of CRFs was by Sha and Pereira [2003], who matched state-of-the-art performance on segmenting noun phrases in text. Since then, linear-chain CRFs have been applied to many problems in natural language processing, including named-entity recognition [McCallum and Li, 2003], feature induction for NER [McCallum, 2003], identifying protein names in biology abstracts [Settles, 2005], segmenting addresses in Web pages [Culotta et al., 2004], finding semantic roles in text [Roth and Yih, 2005], identifying the sources of opinions [Choi et al., 2005], Chinese word segmentation [Peng et al., 2004], Japanese morphological analysis [Kudo et al., 2004], and many others.

In bioinformatics, CRFs have been applied to RNA structural alignment [Sato and Sakakibara, 2005] and protein structure prediction [Liu et al., 2005]. Semi-Markov CRFs [Sarawagi and Cohen, 2005] add somewhat more flexibility in choosing features, which may be useful for certain tasks in information extraction and especially bioinformatics.

General CRFs have also been applied to several tasks in NLP. One promising application is to performing multiple labeling tasks simultaneously. For example, Sutton et al. [2004] show that a two-level dynamic CRF for part-of-speech tagging and noun-phrase chunking performs better than solving the tasks one at a time. Another application is to *multi-label classification*, in which each instance can

have multiple class labels. Rather than learning an independent classifier for each category, Ghamrawi and McCallum [2005] present a CRF that learns dependencies between the categories, resulting in improved classification performance. Finally, the skip-chain CRF, which we present in Section 1.5, is a general CRF that represents long-distance dependencies in information extraction.

An interesting graphical CRF structure has been applied to the problem of proper-noun coreference, that is, of determining which mentions in a document, such as *Mr. President* and *he*, refer to the same underlying entity. McCallum and Wellner [2005] learn a distance metric between mentions using a fully-connected conditional random field in which inference corresponds to graph partitioning. A similar model has been used to segment handwritten characters and diagrams [Cowans and Szummer, 2005, Qi et al., 2005].

In some applications of CRFs, efficient dynamic programs exist even though the graphical model is difficult to specify. For example, McCallum et al. [2005] learn the parameters of a string-edit model in order to discriminate between matching and nonmatching pairs of strings. Also, there is work on using CRFs to learn distributions over the derivations of a grammar [Riezler et al., 2002, Clark and Curran, 2004, Sutton, 2004, Viola and Narasimhan, 2005]. A potentially useful unifying framework for this type of model is provided by case-factor diagrams [McAllester et al., 2004].

In copmputer vision, several authors have used grid-shaped CRFs [He et al., 2004, Kumar and Hebert, 2003] for labeling and segmenting images. Also, for recognizing objects, Quattoni et al. [2005] use a tree-shaped CRF in which latent variables are designed to recognize characteristic parts of an object.

1.4.3 Parameter Estimation

Parameter estimation for general CRFs is essentially the same as for linear-chains, except that computing the model expectations requires more general inference algorithms. First, we discuss the fully-observed case, in which the training and testing data are independent, and the training data is fully observed. In this case the conditional log likelihood is given by

$$\ell(\theta) = \sum_{C_p \in \mathcal{C}} \sum_{\Psi_c \in C_p} \sum_{k=1}^{K(p)} \lambda_{pk} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \log Z(\mathbf{x}). \tag{1.42}$$

It is worth noting that the equations in this section do not explicitly sum over training instances, because if a particular application happens to have iid training instances, they can be represented by disconnected components in the graph G.

The partial derivative of the log likelihood with respect to a parameter λ_{pk} associated with a clique template C_p is

$$\frac{\partial \ell}{\partial \lambda_{pk}} = \sum_{\Psi_c \in C_p} f_{pk}(\mathbf{x}_c, \mathbf{y}_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{y}'_c} f_{pk}(\mathbf{x}_c, \mathbf{y}'_c) p(\mathbf{y}'_c | \mathbf{x}). \tag{1.43}$$

The function $\ell(\theta)$ has many of the same properties as in the linear-chain case. First, the zero-gradient conditions can be interpreted as requiring that the sufficient statistics $F_{pk}(\mathbf{x}, \mathbf{y}) = \sum_{\Psi_c} f_{pk}(\mathbf{x}_c, \mathbf{y}_c)$ have the same expectations under the empirical distribution and under the model distribution. Second, the function $\ell(\theta)$ is concave, and can be efficiently maximized by second-order techniques such as conjugate gradient and L-BFGS. Finally, regularization is used just as in the linear-chain case.

Now, we discuss the case of *within-network classification*, where there are dependencies between the training and testing data. That is, the random variables $\mathbf{y}$ are partitioned into a set $\mathbf{y}^{\mathsf{tr}}$ that is observed during training and a set $\mathbf{y}^{\mathsf{tst}}$ that is unobserved during training. It is assumed that the graph G contains connections between $\mathbf{y}^{\mathsf{tr}}$ and $\mathbf{y}^{\mathsf{tst}}$.

Within-network classification can be viewed as a kind of *latent variable* problem, in which certain variables, in this case $\mathbf{y}^{\mathsf{tst}}$, are not observed in the training data. It is more difficult to train CRFs with latent variables, because optimizing the likelihood $p(\mathbf{y}^{\mathsf{tr}}|\mathbf{x})$ requires marginalizing out the latent variables $\mathbf{y}^{\mathsf{tst}}$. Because of this difficultly, the original work on CRFs focused on fully-observed training data, but recently there has been increasing interest in training latent-variable CRFs [Quattoni et al., 2005, McCallum et al., 2005].

Suppose we have a conditional random field with inputs $\mathbf{x}$ in which the output variables $\mathbf{y}$ are observed in the training data, but we have additional variables $\mathbf{w}$ that are latent, so that the CRF has the form

$$p(\mathbf{y}, \mathbf{w}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p). \tag{1.44}$$

The objective function to maximize during training is the marginal likelihood

$$\ell(\theta) = \log p(\mathbf{y}|\mathbf{x}) = \log \sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x}). \tag{1.45}$$

The first question is how even to compute the marginal likelihood $\ell(\theta)$, because if there are many variables $\mathbf{w}$, the sum cannot be computed directly. The key is to realize that we need to compute $\log \sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x})$ not for any possible assignment $\mathbf{y}$, but only for the particular assignment that occurs in the training data. This motivates taking the original CRF (1.44), and clamping the variables Y to their observed values in the training data, yielding a distribution over $\mathbf{w}$:

$$p(\mathbf{w}|\mathbf{y}, \mathbf{x}) = \frac{1}{Z(\mathbf{y}, \mathbf{x})} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p), \tag{1.46}$$

where the normalization factor is

$$Z(\mathbf{y}, \mathbf{x}) = \sum_{\mathbf{w}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p). \tag{1.47}$$

This new normalization constant $Z(\mathbf{y}, \mathbf{x})$ can be computed by the same inference

algorithm that we use to compute $Z(\mathbf{x})$. In fact, $Z(\mathbf{y}, \mathbf{x})$ is easier to compute, because it sums only over $\mathbf{w}$, while $Z(\mathbf{x})$ sums over both $\mathbf{w}$ and $\mathbf{y}$. Graphically, this amounts to saying that clamping the variables $\mathbf{y}$ in the graph G can simplify the structure among $\mathbf{w}$.

Once we have $Z(\mathbf{y}, \mathbf{x})$, the marginal likelihood can be computed as

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \sum_{\mathbf{w}} \prod_{C_p \in \mathcal{C}} \prod_{\Psi_c \in C_p} \Psi_c(\mathbf{x}_c, \mathbf{w}_c, \mathbf{y}_c; \theta_p) = \frac{Z(\mathbf{y}, \mathbf{x})}{Z(\mathbf{x})}. \tag{1.48}$$

Now that we have a way to compute ℓ, we discuss how to maximize it with respect to θ. Maximizing $\ell(\theta)$ can be difficult because ℓ is no longer convex in general (intuitively, log-sum-exp is convex, but the difference of two log-sum-exp functions might not be), so optimization procedures are typically guaranteed to find only local maxima. Whatever optimization technique is used, the model parameters must be carefully initialized in order to reach a good local maximum.

We discuss two different ways to maximize ℓ: directly using the gradient, as in Quattoni et al. [2005]; and using EM, as in McCallum et al. [2005]. To maximize ℓ directly, we need to calculate its gradient. The simplest way to do this is to use the following fact. For any function $f(\lambda)$, we have

$$\frac{df}{d\lambda} = f(\lambda) \frac{d \log f}{d\lambda}, \tag{1.49}$$

which can be seen by applying the chain rule to $\log f$ and rearranging. Applying this to the marginal likelihood $\ell(\Lambda) = \log \sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x})$ yields

$$\frac{\partial \ell}{\partial \lambda_{pk}} = \frac{1}{\sum_{\mathbf{w}} p(\mathbf{y}, \mathbf{w}|\mathbf{x})} \sum_{\mathbf{w}} \frac{\partial}{\partial \lambda_{pk}} \big[p(\mathbf{y}, \mathbf{w}|\mathbf{x})\big] \tag{1.50}$$

$$= \sum_{\mathbf{w}} p(\mathbf{w}|\mathbf{y}, \mathbf{x}) \frac{\partial}{\partial \lambda_{pk}} \big[\log p(\mathbf{y}, \mathbf{w}|\mathbf{x})\big]. \tag{1.51}$$

This is the expectation of the fully-observed gradient, where the expectation is taken over $\mathbf{w}$. This expression simplifies to

$$\frac{\partial \ell}{\partial \lambda_{pk}} = \sum_{\Psi_c \in C_p} \sum_{\mathbf{w}'_c} p(\mathbf{w}'_c|\mathbf{y}, \mathbf{x}) f_k(\mathbf{y}_c, \mathbf{x}_c, \mathbf{w}'_c) - \sum_{\Psi_c \in C_p} \sum_{\mathbf{w}'_c, \mathbf{y}'_c} p(\mathbf{w}'_c, \mathbf{y}'_c|\mathbf{x}_c) f_k(\mathbf{y}'_c, \mathbf{x}_c, \mathbf{w}'_c). \tag{1.52}$$

This gradient requires computing two different kinds of marginal probabilities. The first term contains a marginal probability $p(\mathbf{w}'_c|\mathbf{y}, \mathbf{x})$, which is exactly a marginal distribution of the clamped CRF (1.46). The second term contains a different marginal $p(\mathbf{w}'_c, \mathbf{y}'_c|\mathbf{x}_c)$, which is the same marginal probability required in a fully-observed CRF. Once we have computed the gradient, ℓ can be maximized by standard techniques such as conjugate gradient. In our experience, conjugate gradient tolerates violations of convexity better than limited-memory BFGS, so it may be a better choice for latent-variable CRFs.

Alternatively, ℓ can be optimized using expectation maximization (EM). At each

iteration j in the EM algorithm, the current parameter vector $\theta^{(j)}$ is updated as follows. First, in the E-step, an auxiliary function $q(\mathbf{w})$ is computed as $q(\mathbf{w}) = p(\mathbf{w}|\mathbf{y}, \mathbf{x}; \theta^{(j)})$. Second, in the M-step, a new parameter vector $\theta^{(j+1)}$ is chosen as

$$\theta^{(j+1)} = \arg\max_{\theta'} \sum_{\mathbf{w}'} q(\mathbf{w}') \log p(\mathbf{y}, \mathbf{w}'|\mathbf{x}; \theta'). \tag{1.53}$$

The direct maximization algorithm and the EM algorithm are strikingly similar. This can be seen by substituting the definition of q into (1.53) and taking derivatives. The gradient is almost identical to the direct gradient (1.52). The only difference is that in EM, the distribution $p(\mathbf{w}|\mathbf{y}, \mathbf{x})$ is obtained from a previous, fixed parameter setting rather than from the argument of the maximization. We are unaware of any empirical comparison of EM to direct optimization for latent-variable CRFs.

1.4.4 Inference

In general CRFs, just as in the linear-chain case, gradient-based training requires computing marginal distributions $p(\mathbf{y}_c|\mathbf{x})$, and testing requires computing the most likely assignment $\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x})$. This can be accomplished using any inference algorithm for graphical models. If the graph has small treewidth, then the junction tree algorithm can be used to exactly compute the marginals, but because both inference problems are NP-hard for general graphs, this is not always possible. In such cases, approximate inference must be used to compute the gradient. In this section, we mention various approximate inference algorithms that have been used successfully with CRFs. Detailed discussion of these are beyond the scope of this tutorial.

When choosing an inference algorithm to use within CRF training, the important thing to understand is that it will be invoked repeatedly, once for each time that the gradient is computed. For this reason, sampling-based approaches which may take many iterations to converge, such as Markov chain Monte Carlo, have not been popular, although they might be appropriate in some circumstances. Indeed, contrastive divergence [Hinton, 2000], in which an MCMC sampler is run for only a few samples, has been successfully applied to CRFs in vision [He et al., 2004].

Because of their computational efficiency, variational approaches have been most popular for CRFs. Several authors [Taskar et al., 2002, Sutton et al., 2004] have used loopy belief propagation. Belief propagation is an exact inference algorithm for trees which generalizes the forward-backward. Although the generalization of the forward-backward recursions, which are called *message updates*, are neither exact nor even guaranteed to converge if the model is not a tree, they are still well-defined, and they have been empirically successful in a wide variety of domains, including text processing, vision, and error-correcting codes. In the past five years, there has been much theoretical analysis of the algorithm as well. We refer the reader to Yedidia et al. [2004] for more information.

1.4.5 Discussion

This section contains miscellaneous remarks about CRFs. First, it is easily seen that logistic regression model (1.7) is a conditional random field with a single output variable. Thus, CRFs can be viewed as an extension of logistic regression to arbitrary graphical structures.

Although we have emphasized the view of a CRF as a model of the conditional distribution, one could view it as an objective function for parameter estimation of joint distributions. As such, it is one objective among many, including generative likelihood, pseudolikelihood [Besag, 1977], and the maximum-margin objective [Taskar et al., 2004, Altun et al., 2003]. Another related discriminative technique for structured models is the averaged perceptron, which has been especially popular in the natural language community [Collins, 2002], in large part because of its ease of implementation. To date, there has been little careful comparison of these, especially CRFs and max-margin approaches, across different structures and domains.

Given this view, it is natural to imagine training directed models by conditional likelihood, and in fact this is commonly done in the speech community, where it is called maximum mutual information training. However, it is no easier to maximize the conditional likelihood in a directed model than an undirected model, because in a directed model the conditional likelihood requires computing $\log p(\mathbf{x})$, which plays the same role as $Z(\mathbf{x})$ in the CRF likelihood. In fact, training is more complex in a directed model, because the model parameters are constrained to be probabilities—constraints which can make the optimization problem more difficult. This is in stark contrast to the joint likelihood, which is much easier to compute for directed models than undirected models (although recently several efficient parameter estimation techniques have been proposed for undirected factor graphs, such as Abbeel et al. [2005] and Wainwright et al. [2003]).

1.4.6 Implementation Concerns

There are a few implementation techniques that can help both training time and accuracy of CRFs, but are not always fully discussed in the literature. Although these apply especially to language applications, they are also useful more generally. First, when the predicted variables are discrete, the features f_{pk} are ordinarily chosen to have a particular form:

$$f_{pk}(\mathbf{y}_c, \mathbf{x}_c) = \mathbf{1}_{\{\mathbf{y}_c = \tilde{\mathbf{y}}_c\}} q_{pk}(\mathbf{x}_c). \tag{1.54}$$

In other words, each feature is nonzero only for a single output configuration $\tilde{\mathbf{y}}_c$, but as long as that constraint is met, then the feature value depends only on the input observation. Essentially, this means that we can think of our features as depending only on the input $\mathbf{x}_c$, but that we have a separate set of weights for each output configuration. This feature representation is also computationally efficient, because computing each q_{pk} may involve nontrivial text or image processing, and it need be

evaluated only once for every feature that uses it. To avoid confusion, we refer to the functions $q_{pk}(\mathbf{x}_c)$ as *observation functions* rather than as features. Examples of observation functions are "word x_t is capitalized" and "word x_t ends in *ing*".

This representation can lead to a large number of features, which can have significant memory and time requirements. For example, to match state-of-the-art results on a standard natural language task, Sha and Pereira [2003] use 3.8 million features. Not all of these features are ever nonzero in the training data. In particular, some observation functions q_{pk} are nonzero only for certain output configurations. This point can be confusing: One might think that such features can have no effect on the likelihood, but actually they do affect $Z(\mathbf{x})$, so putting a negative weight on them can improve the likelihood by making wrong answers less likely. In order to save memory, however, sometimes these *unsupported features*, that is, those which never occur in the training data, are removed from the model. In practice, however, including unsupported features typically results in better accuracy.

In order to get the benefits of unsupported features with less memory, we have had success with an ad hoc technique for selecting only a few unsupported features. The main idea is to add unsupported features only for likely paths, as follows: first train a CRF without any unsupported features, stopping after only a few iterations; then add unsupported features $f_{pk}(\mathbf{y}_c, \mathbf{x}_c)$ for cases where $\mathbf{x}_c$ occurs in the training data, and $p(\mathbf{y}_c|\mathbf{x}) > \epsilon$. McCallum [2003] presents a more principled method of feature selection for CRFs.

Second, if the observations are categorical rather than ordinal, that is, if they are discrete but have no intrinsic order, it is important to convert them to binary features. For example, it makes sense to learn a linear weight on $f_k(y, x_t)$ when f_k is 1 if x_t is the word *dog* and 0 otherwise, but not when f_k is the integer index of word x_t in the text's vocabulary. Thus, in text applications, CRF features are typically binary; in other application areas, such as vision and speech, they are more commonly real-valued.

Third, in language applications, it is sometimes helpful to include redundant factors in the model. For example, in a linear-chain CRF, one may choose to include both edge factors $\Psi_t(y_t, y_{t-1}, \mathbf{x}_t)$ and variable factors $\Psi_t(y_t, \mathbf{x}_t)$. Although one could define the same family of distributions using only edge factors, the redundant node factors provide a kind of backoff, which is useful when there is too little data. In language applications, there is always too little data, even when hundreds of thousands of words are available.

Finally, often the probabilities involved in forward-backward and belief propagation become too small to be represented within numerical precision. There are two standard approaches to this common problem. One approach is to normalize each of the vectors α_t and β_t to sum to 1, thereby magnifying small values. A second approach is to perform computations in the logarithmic domain, e.g., the forward recursion becomes

$$\log \alpha_t(j) = \bigoplus_{i \in S} \big(\log \Psi_t(j, i, x_t) + \log \alpha_{t-1}(i) \big), \tag{1.55}$$

where $\oplus$ is the operator $a \oplus b = \log(e^a + e^b)$. At first, this does not seem much of an improvement, since numerical precision is lost when computing e^a and e^b. But $\oplus$ can be computed as

$$a \oplus b = a + \log(1 + e^{b-a}) = b + \log(1 + e^{a-b}), \tag{1.56}$$

which can be much more numerically stable, particularly if we pick the version of the identity with the smaller exponent. CRF implementations often use the log-space approach because it makes computing $Z(\mathbf{x})$ more convenient, but in some applications, the computational expense of taking logarithms is an issue, making normalization preferable.

1.5 Skip-Chain CRFs

In this section, we present a case study of applying a general CRF to a practical natural language problem. In particular, we consider a problem in *information extraction*, the task of building a database automatically from unstructured text. Recent work in extraction has often used sequence models, such as HMMs and linear-chain CRFs, which model dependencies only between neighboring labels, on the assumption that those dependencies are the strongest.

But sometimes it is important to model certain kinds of long-range dependencies between entities. One important kind of dependency within information extraction occurs on repeated mentions of the same field. When the same entity is mentioned more than once in a document, such as *Robert Booth*, in many cases all mentions have the same label, such as SEMINAR-SPEAKER. We can take advantage of this fact by favoring labelings that treat repeated words identically, and by combining features from all occurrences so that the extraction decision can be made based on global information. Furthermore, identifying all mentions of an entity can be useful in itself, because each mention might contain different useful information. However, most extraction systems, whether probabilistic or not, do not take advantage of this dependency, instead treating the separate mentions independently.

To perform collective labeling, we need to represent dependencies between distant terms in the input. But this reveals a general limitation of sequence models, whether generatively or discriminatively trained. Sequence models make a Markov assumption among labels, that is, that any label y_t is independent of all previous labels given its immediate predecessors $y_{t-k} \dots y_{t-1}$. This represents dependence only between nearby nodes—for example, between bigrams and trigrams—and cannot represent the higher-order dependencies that arise when identical words occur throughout a document.

To relax this assumption, we introduce the *skip-chain CRF*, a conditional model that collectively segments a document into mentions and classifies the mentions by entity type, while taking into account probabilistic dependencies between distant mentions. These dependencies are represented in a skip-chain model by augmenting

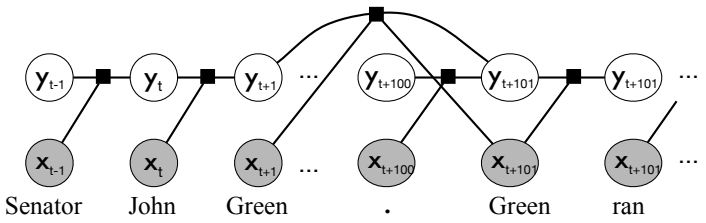

Figure 1.5 Graphical representation of a skip-chain CRF. Identical words are connected because they are likely to have the same label.

$w_t = w$
w_t matches `[A-Z][a-z]+`
w_t matches `[A-Z][A-Z]+`
w_t matches `[A-Z]`
w_t matches `[A-Z]+`
w_t matches `[A-Z]+[a-z]+[A-Z]+[a-z]`
w_t appears in list of first names, last names, honorifics, etc.
w_t appears to be part of a time followed by a dash
w_t appears to be part of a time preceded by a dash
w_t appears to be part of a date
$T_t = T$
$q_k(\mathbf{x}, t+\delta)$ for all k and $\delta \in [-4, 4]$

Table 1.1 Input features $q_k(\mathbf{x}, t)$ for the seminars data. In the above w_t is the word at position t, T_t is the POS tag at position t, w ranges over all words in the training data, and T ranges over all part-of-speech tags returned by the Brill tagger. The "appears to be" features are based on hand-designed regular expressions that can span several tokens.

a linear-chain CRF with factors that depend on the labels of distant but similar words. This is shown graphically in Figure 1.5.

Even though the limitations of n-gram models have been widely recognized within natural language processing, long-distance dependencies are difficult to represent in generative models, because full n-gram models have too many parameters if n is large. We avoid this problem by selecting which skip edges to include based on the input string. This kind of input-specific dependence is difficult to represent in a generative model, because it makes generating the input more complicated. In other words, conditional models have been popular because of their flexibility in allowing overlapping features; skip-chain CRFs take advantage of their flexibility in allowing input-specific model structure.

1.5.1 Model

The skip-chain CRF is essentially a linear-chain CRF with additional long-distance edges between similar words. We call these additional edges *skip edges*. The features on skip edges can incorporate information from the context of both endpoints, so that strong evidence at one endpoint can influence the label at the other endpoint. When applying the skip-chain model, we must choose which skip edges to include. The simplest choice is to connect all pairs of identical words, but more generally we can connect any pair of words that we believe to be similar, for example, pairs of words that belong to the same stem class, or have small edit distance. In addition, we must be careful not to include too many skip edges, because this could result in a graph that makes approximate inference difficult. So we need to use similarity metrics that result in a sufficiently sparse graph. In the experiments below, we focus on named-entity recognition, so we connect pairs of identical capitalized words.

Formally, the skip-chain CRF is defined as a general CRF with two clique templates: one for the linear-chain portion, and one for the skip edges. For an sentence $\mathbf{x}$, let $\mathcal{I} = \{(u, v)\}$ be the set of all pairs of sequence positions for which there are skip edges. For example, in the experiments reported here, $\mathcal{I}$ is the set of indices of all pairs of identical capitalized words. Then the probability of a label sequence $\mathbf{y}$ given an input $\mathbf{x}$ is modeled as

$$p_\theta(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{t=1}^{T} \Psi_t(y_t, y_{t-1}, \mathbf{x}) \prod_{(u,v)\in\mathcal{I}} \Psi_{uv}(y_u, y_v, \mathbf{x}), \tag{1.57}$$

where Ψ_t are the factors for linear-chain edges, and Ψ_{uv} are the factors over skip edges. These factors are defined as

$$\Psi_t(y_t, y_{t-1}, \mathbf{x}) = \exp\left\{\sum_k \lambda_{1k} f_{1k}(y_t, y_{t-1}, \mathbf{x}, t)\right\} \tag{1.58}$$

$$\Psi_{uv}(y_u, y_v, \mathbf{x}) = \exp\left\{\sum_k \lambda_{2k} f_{2k}(y_u, y_v, \mathbf{x}, u, v)\right\}, \tag{1.59}$$

where $\theta_1 = \{\lambda_{1k}\}_{k=1}^{K_1}$ are the parameters of the linear-chain template, and $\theta_2 = \{\lambda_{2k}\}_{k=1}^{K_2}$ are the parameters of the skip template. The full set of model parameters are $\theta = \{\theta_1, \theta_2\}$.

As described in Section 1.4.6, both the linear-chain features and skip-chain features are factorized into indicator functions of the outputs and observation functions, as in (1.54). In general the observation functions $q_k(\mathbf{x}, t)$ can depend on arbitrary positions of the input string. For example, a useful feature for NER is $q_k(\mathbf{x}, t) = 1$ if and only if x_{t+1} is a capitalized word.

System	stime	etime	location	speaker	overall
BIEN Peshkin and Pfeffer [2003]	96.0	**98.8**	87.1	76.9	89.7
Linear-chain CRF	**97.5**	97.5	**88.3**	77.3	90.2
Skip-chain CRF	96.7	97.2	88.1	**80.4**	**90.6**

Table 1.2 Comparison of F_1 performance on the seminars data. The top line gives a dynamic Bayes net that has been previously used on this data set. The skip-chain CRF beats the previous systems in overall F1 and on the speaker field, which has proved to be the hardest field of the four. Overall F1 is the average of the F1 scores for the four fields.

The observation functions for the skip edges are chosen to combine the observations from each endpoint. Formally, we define the feature functions for the skip edges to factorize as:

$$f'_k(y_u, y_v, \mathbf{x}, u, v) = \mathbf{1}_{\{y_u = \tilde{y}_u\}} \mathbf{1}_{\{y_v = \tilde{y}_v\}} q'_k(\mathbf{x}, u, v) \tag{1.60}$$

This choice allows the observation functions $q'_k(\mathbf{x}, u, v)$ to combine information from the neighborhood of y_u and y_v. For example, one useful feature is $q'_k(\mathbf{x}, u, v) = 1$ if and only if $x_u = x_v =$ "Booth" and $x_{v-1} =$ "Speaker:". This can be a useful feature if the context around x_u, such as "Robert Booth is manager of control engineering...," may not make clear whether or not Robert Booth is presenting a talk, but the context around x_v is clear, such as "Speaker: Robert Booth." [1]
Because the loops in a skip-chain CRF can be long and overlapping, exact inference is intractable for the data we consider. The running time required by exact inference is exponential in the size of the largest clique in the graph's junction tree. In junction trees created from the seminars data, 29 of the 485 instances have a maximum clique size of 10 or greater, and 11 have a maximum clique size of 14 or greater. (The worst instance has a clique with 61 nodes.) These cliques are far too large to perform inference exactly. For reference, representing a single factor that depends on 14 variables requires more memory than can be addressed in a 32-bit architecture. Instead, we perform approximate inference using loopy belief propagation, which was mentioned in Section 1.4.4. We use an asynchronous tree-based schedule known as TRP [Wainwright et al., 2001].

1.5.2 Results

We evaluate skip-chain CRFs on a collection of 485 e-mail messages announcing seminars at Carnegie Mellon University. The messages are annotated with the seminar's starting time, ending time, location, and speaker. This data set is due to

1. This example is taken from an actual error made by a linear-chain CRF on the seminars data set. We present results from this data set in Section 1.5.2.

Field	Linear-chain	Skip-chain
stime	12.6	17
etime	3.2	5.2
location	6.4	0.6
speaker	30.2	4.8

Table 1.3 Number of inconsistently mislabeled tokens, that is, tokens that are mislabeled even though the same token is labeled correctly elsewhere in the document. Learning long-distance dependencies reduces this kind of error in the speaker and location fields. Numbers are averaged over 5 folds.

Freitag [1998], and has been used in much previous work.
Often the fields are listed multiple times in the message. For example, the speaker name might be included both near the beginning and later on, in a sentence like "If you would like to meet with Professor Smith..." As mentioned earlier, it can be useful to find both such mentions, because different information can occur in the surrounding context of each mention: for example, the first mention might be near an institutional affiliation, while the second mentions that Smith is a professor.
We evaluate a skip-chain CRF with skip edges between identical capitalized words. The motivation for this is that the hardest aspect of this data set is identifying speakers and locations, and capitalized words that occur multiple times in a seminar announcement are likely to be either speakers or locations.
Table 1.1 shows the list of input features we used. For a skip edge (u, v), the input features we used were the disjunction of the input features at u and v, that is,

$$q'_k(\mathbf{x}, u, v) = q_k(\mathbf{x}, u) \oplus q_k(\mathbf{x}, v) \tag{1.61}$$

where $\oplus$ is binary or. All of our results are averaged over 5-fold cross-validation with an 80/20 split of the data. We report results from both a linear-chain CRF and a skip-chain CRF with the same set of input features.
We calculate precision and recall as[2]

$$P = \frac{\#\text{ tokens extracted correctly}}{\#\text{ tokens extracted}}$$
$$R = \frac{\#\text{ tokens extracted correctly}}{\#\text{ true tokens of field}}$$

2. Previous work on this data set has traditionally measured precision and recall per document, that is, from each document the system extracts only one field of each type. Because the goal of the skip-chain CRF is to extract all mentions in a document, these metrics are inappropriate, so we cannot compare with this previous work. Peshkin and Pfeffer [2003] do use the per-token metric (personal communication), so our comparison is fair in that respect.

As usual, we report $F_1 = (2PR)/(P + R)$.

Table 1.2 compares a skip-chain CRF to a linear-chain CRF and to a dynamic Bayes net used in previous work [Peshkin and Pfeffer, 2003]. The skip-chain CRF performs much better than all the other systems on the SPEAKER field, which is the field for which the skip edges would be expected to make the most difference. On the other fields, however, the skip-chain CRF does slightly worse (less than 1% absolute F1).

We expected that the skip-chain CRF would do especially well on the speaker field, because speaker names tend to appear multiple times in a document, and a skip-chain CRF can learn to label the multiple occurrences consistently. To test this hypothesis, we measure the number of *inconsistently mislabeled* tokens, that is, tokens that are mislabeled even though the same token is classified correctly elsewhere in the document. Table 1.3 compares the number of inconsistently mislabeled tokens in the test set between linear-chain and skip-chain CRFs. For the linear-chain CRF, on average 30.2 true speaker tokens are inconsistently mislabeled. Because the linear-chain CRF mislabels 121.6 true speaker tokens, this situation includes 24.7% of the missed speaker tokens.

The skip-chain CRF shows a dramatic decrease in inconsistently mislabeled tokens on the speaker field, from 30.2 tokens to 4.8. Consequently, the skip-chain CRF also has much better recall on speaker tokens than the linear-chain CRF (70.0 R linear chain, 76.8 R skip chain). This explains the increase in F1 from linear-chain to skip-chain CRFs, because the two have similar precision (86.5 P linear chain, 85.1 skip chain). These results support the original hypothesis that treating repeated tokens consistently especially benefits recall on the SPEAKER field.

On the LOCATION field, on the other hand, where we might also expect skip-chain CRFs to perform better, there is no benefit. We explain this by observing in Table 1.3 that inconsistent misclassification occurs much less frequently in this field.

1.5.3 Related Work

Recently, Bunescu and Mooney [2004] have used a relational Markov network to collectively classify the mentions in a document, achieving increased accuracy by learning dependencies between similar mentions. In their work, however, candidate phrases are extracted heuristically, which can introduce errors if a true entity is not selected as a candidate phrase. Our model performs collective segmentation and labeling simultaneously, so that the system can take into account dependencies between the two tasks.

As an extension to our work, Finkel et al. [2005] augment the skip-chain model with richer kinds of long-distance factors than just over pairs of words. These factors are useful for modeling exceptions to the assumption that similar words tend to have similar labels. For example, in named-entity recognition, the word *China* is as a place name when it appears alone, but when it occurs within the phrase *The China Daily*, it should be labeled as a organization. Because this model is more

complex than the original skip-chain model, Finkel et al. estimate its parameters in two stages, first training the linear-chain component as a separate CRF, and then heuristically selecting parameters for the long-distance factors. Finkel et al. report improved results both on the seminars data set that we consider in this chapter, and on several other standard information extraction data sets.

Finally, the skip-chain CRF can also be viewed as performing extraction while taking into account a simple form of coreference information, since the reason that identical words are likely to have similar tags is that they are likely to be coreferent. Thus, this model is a step toward joint probabilistic models for extraction and data mining as advocated by McCallum and Jensen [2003]. An example of such a joint model is the one of Wellner et al. [2004], which jointly segments citations in research papers and predicts which citations refer to the same paper.

1.6 Conclusion

Conditional random fields are a natural choice for many relational problems because they allow both graphically representing dependencies between entities, and including rich observed features of entities. In this chapter, we have presented a tutorial on CRFs, covering both linear-chain models and general graphical structures. Also, as a case study in CRFs for collective classification, we have presented the skip-chain CRF, a type of general CRF that performs joint segmentation and collective labeling on a practical language understanding task.

The main disadvantage of CRFs is the computational expense of training. Although CRF training is feasible for many real-world problems, the need to perform inference repeatedly during training becomes a computational burden when there are a large number of training instances, when the graphical structure is complex, when there are latent variables, or when the output variables have many outcomes. One focus of current research [Abbeel et al., 2005, Sutton and McCallum, 2005, Wainwright et al., 2003] is on more efficient parameter estimation techniques.

Acknowledgments

We thank Tom Minka and Jerod Weinman for helpful conversations, and we thank Francine Chen and Benson Limketkai for useful comments. This work was supported in part by the Center for Intelligent Information Retrieval; in part by the Defense Advanced Research Projects Agency (DARPA), the Department of the Interior, NBC, Acquisition Services Division, under contract number NBCHD030010; and in part by The Central Intelligence Agency, the National Security Agency and National Science Foundation under NSF grants #IIS-0427594 and #IIS-0326249. Any opinions, findings and conclusions or recommendations expressed in this material are the author(s) and do not necessarily reflect those of the sponsors.

References

Pieter Abbeel, Daphne Koller, and Andrew Y. Ng. Learning factor graphs in polynomial time and sample complexity. In *Twenty-first Conference on Uncertainty in Artificial Intelligence (UAI05)*, 2005.

Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. Hidden Markov support vector machines. In *20th International Conference on Machine Learning (ICML)*, 2003.

Dimitri P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2nd edition, 1999.

Julian Besag. Efficiency of pseudolikelihood estimation for simple gaussian fields. *Biometrika*, 64(3):616–618, 1977.

Razvan Bunescu and Raymond J. Mooney. Collective information extraction with relational Markov networks. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics*, 2004.

Richard H. Byrd, Jorge Nocedal, and Robert B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Math. Program.*, 63 (2):129–156, 1994.

Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms using different performance metrics. Technical Report TR2005-1973, Cornell University, 2005. Available at `http://www.cs.cornell.edu/~alexn/`.

Yejin Choi, Claire Cardie, Ellen Riloff, and Siddharth Patwardhan. Identifying sources of opinions with conditional random fields and extraction patterns. In *Proceedings of the Human Language Technology Conference/Conference on Empirical Methods in Natural Language Processing (HLT-EMNLP)*, 2005.

Stephen Clark and James R. Curran. Parsing the WSJ using CCG and log-linear models. In *Proceedings of the 42nd Meeting of the Association for Computational Linguistics (ACL'04), Main Volume*, pages 103–110, Barcelona, Spain, July 2004.

Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2002.

Philip J. Cowans and Martin Szummer. A graphical model for simultaneous partitioning and labeling. In *Tenth International Workshop on Artificial Intelligence and Statistics*, 2005.

Aron Culotta, Ron Bekkerman, and Andrew McCallum. Extracting social networks and contact information from email and the web. In *First Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, 2004.

Aron Culotta and Andrew McCallum. Confidence estimation for information extraction. In *Human Language Technology Conference (HLT)*, 2004.

Jenny Finkel, Trond Grenager, and Christopher D. Manning. Incorporating non-local information into information extraction systems by Gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL)*, 2005.

Dayne Freitag. *Machine Learning for Information Extraction in Informal Domains.* PhD thesis, Carnegie Mellon University, 1998.

Nadia Ghamrawi and Andrew McCallum. Collective multi-label classification. In *Conference on Information and Knowledge Management (CIKM)*, 2005.

Joshua Goodman. Exponential priors for maximum entropy models. In *Proceedings of the Human Language Technology Conference/North American Chapter of the Association for Computational Linguistics (HLT/NAACL)*, 2004.

Xuming He, Richard S. Zemel, and Miguel Á. Carreira-Perpiñián. Multiscale conditional random fields for image labelling. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2004.

G.E. Hinton. Training products of experts by minimizing contrastive divergence. Technical Report 2000-004, Gatsby Computational Neuroscience Unit, 2000.

F. R. Kschischang, B. J. Frey, and H. A. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

Taku Kudo, Kaoru Yamamoto, and Yuji Matsumoto. Applying conditional random fields to Japanese morphological analysis. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2004.

Sanjiv Kumar and Martial Hebert. Discriminative fields for modeling spatial dependencies in natural images. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2003.

J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. *Proc. 18th International Conf. on Machine Learning*, 2001.

Yan Liu, Jaime Carbonell, Peter Weigele, and Vanathi Gopalakrishnan. Segmentation conditional random fields (SCRFs): A new approach for protein fold recognition. In *ACM International conference on Research in Computational Molecular Biology (RECOMB05)*, 2005.

R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In Dan Roth and Antal van den Bosch, editors, *Proceedings of the Sixth Conference on Natural Language Learning (CoNLL-2002)*, pages 49–55, 2002.

David McAllester, Michael Collins, and Fernando Pereira. Case-factor diagrams for structured probabilistic modeling. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

Andrew McCallum. Efficiently inducing features of conditional random fields. In *Conference on Uncertainty in AI (UAI)*, 2003.

Andrew McCallum, Kedar Bellare, and Fernando Pereira. A conditional random field for discriminatively-trained finite-state string edit distance. In *Conference on Uncertainty in AI (UAI)*, 2005.

Andrew McCallum and David Jensen. A note on the unification of information extraction and data mining using conditional-probability, relational models. In *IJCAI'03 Workshop on Learning Statistical Models from Relational Data*, 2003.

Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Seventh Conference on Natural Language Learning (CoNLL)*, 2003.

Andrew McCallum and Ben Wellner. Conditional models of identity uncertainty with application to noun coreference. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 905–912. MIT Press, Cambridge, MA, 2005.

Thomas P. Minka. A comparsion of numerical optimizers for logistic regression. Technical report, 2003. `http://research.microsoft.com/~minka/papers/logreg/`.

Tom Minka. Discriminative models, not discriminative training. Technical Report MSR-TR-2005-144, Microsoft Research, October 2005. `ftp://ftp.research.microsoft.com/ pub/tr/TR-2005-144.pdf` .

A. Y. Ng and M. I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, pages 841–848, Cambridge, MA, 2002. MIT Press.

Fuchun Peng, Fangfang Feng, and Andrew McCallum. Chinese segmentation and new word detection using conditional random fields. In *Proceedings of The 20th International Conference on Computational Linguistics (COLING)*, pages 562–568, 2004.

Fuchun Peng and Andrew McCallum. Accurate information extraction from research papers using conditional random fields. In *Proceedings of Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT-NAACL'04)*, 2004.

Leonid Peshkin and Avi Pfeffer. Bayesian information extraction network. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2003.

Yuan Qi, Martin Szummer, and Thomas P. Minka. Diagram structure recognition by Bayesian conditional random fields. In *International Conference on Computer Vision and Pattern Recognition*, 2005.

Ariadna Quattoni, Michael Collins, and Trevor Darrell. Conditional random fields for object recognition. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1097–1104. MIT Press, Cambridge, MA, 2005.

L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257 – 286, 1989.

Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 2005. To appear.

S. Riezler, T. King, R. Kaplan, R. Crouch, J. Maxwell, and M. Johnson. Parsing the Wall Street Journal using a lexical-functional grammar and discriminative estimation techniques. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002.

D. Roth and W. Yih. Integer linear programming inference for conditional random fields. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 737–744, 2005.

Sunita Sarawagi and William W. Cohen. Semi-Markov conditional random fields for information extraction. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 1185–1192. MIT Press, Cambridge, MA, 2005.

Kengo Sato and Yasubumi Sakakibara. RNA secondary structural alignment with conditional random fields. *Bioinformatics*, 21:ii237–242, 2005.

Burr Settles. Abner: an open source tool for automatically tagging genes, proteins, and other entity names in text. *Bioinformatics*, 21(14):3191–3192, 2005.

Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL*, pages 213–220, 2003.

P. Singla and P. Domingos. Discriminative training of Markov logic networks. In *Proceedings of the Twentieth National Conference on Artificial Intelligence*, pages 868–873, Pittsburgh, PA, 2005. AAAI Press.

Charles Sutton. Conditional probabilistic context-free grammars. Master's thesis, University of Massachusetts, 2004. `http://www.cs.umass.edu/~casutton/publications.html`.

Charles Sutton and Andrew McCallum. Piecewise training of undirected models. In *21st Conference on Uncertainty in Artificial Intelligence*, 2005.

Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML)*, 2004.

Ben Taskar, Pieter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI02)*, 2002.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks.

In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*. MIT Press, Cambridge, MA, 2004.

Paul Viola and Mukund Narasimhan. Learning to extract information from semi-structured text using a discriminative context free grammar. In *Proceedings of the ACM SIGIR*, 2005.

M. Wainwright, T. Jaakkola, and A. Willsky. Tree-based reparameterization for approximate estimation on graphs with cycles. *Advances in Neural Information Processing Systems (NIPS)*, 2001.

M. J. Wainwright, T. Jaakkola, and A. S. Willsky. Tree-reweighted belief propagation and approximate ML estimation by pseudo-moment matching. In *Ninth Workshop on Artificial Intelligence and Statistics*, 2003.

Hanna Wallach. Efficient training of conditional random fields. M.Sc. thesis, University of Edinburgh, 2002.

Ben Wellner, Andrew McCallum, Fuchun Peng, and Michael Hay. An integrated, conditional model of information extraction and coreference with application to citation graph construction. In *20th Conference on Uncertainty in Artificial Intelligence (UAI)*, 2004.

J.S. Yedidia, W.T. Freeman, and Y. Weiss. Constructing free energy approximations and generalized belief propagation algorithms. Technical Report TR2004-040, Mitsubishi Electric Research Laboratories, 2004.